

# Week 12: Homework 2: Project: Chapter 7: Building Predictive Analytics For AWS IoT

Md Shahadat Hossain Khan (19609)

Github Repository: <https://github.com/mkhan-sfbu/AWS-IoT-Raspberry-Pi-Emulator-Sensor-HAT-Emulator-Predictive-Analysis>

In this homework, we will use machine learning mechanism which is predictive analytics. We are going to use AWS to take a decision based on our existing data. We will retrieve that decision into our Raspberry PI by providing required data from SenseHAT (temperature and humidity) and AWS will tell us if we need watering or not based on our predefined data. To achieve this we will do following tasks -

1. Preparing training data
2. Train AWS AI
3. Prepare AWS to serve data
4. Retrieve predictive data into Raspberry Pi

## Preparing training data

To train AWS AI, we may use any logical data. Here I use data from <https://followpearl.mit.edu/data/> in this page click on "download the latest seven days of data in a JSON format". After downloading it we need to convert this into CSV file. To do this visit <https://data.page/json/csv> and upload our JSON file and it will provide CSV file of that JSON file.

Now there you will find many columns, I just delete all columns but temperature and humidity column. I add another column labeled as "watering" and fill up "watering" column data by "y" or "n" based on some understanding of my own like maintain a ration between temperature and humidity value. Also we need to careful that there must remain NO space between rows and I delete non-numeric or empty row from this CSV file.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
133	2021-11-17 00:00:07	10.9	64.4																		
139	2021-11-17 10:00:07	19.1	46.71	y																	
140	2021-11-17 11:00:07	21.38	43.69	n																	
141	2021-11-17 12:00:07	23.1	40.8	y																	
142	2021-11-17 13:00:07	23.49	39.8	y																	
143	2021-11-17 14:00:08	22.01	43.27	y																	
144	2021-11-17 15:00:07	17.67	51.58	y																	
145	2021-11-17 16:00:07	15.7	51.44	y																	
146	2021-11-17 17:00:07	13.44	52.69	y																	
147	2021-11-17 18:00:07	13.3	50.75	y																	
148	2021-11-17 19:00:07	12.61	56.21	y																	
149	2021-11-17 20:00:07	13.05	56.95	y																	
150	2021-11-17 21:00:07	13.56	56.68	y																	
151	2021-11-17 22:00:07	14.2	55.11	y																	
152	2021-11-17 23:00:08	14.95	54.92	n																	
153	2021-11-18 00:00:07	15.68	54.14	n																	
154	2021-11-18 01:00:07	16.31	53.5	n																	
155	2021-11-18 02:00:08	16.95	52.81	n																	
156	2021-11-18 03:00:07	17.43	52.6	y																	
157	2021-11-18 04:00:07	17.8	52.4	y																	
158	2021-11-18 05:00:07	17.91	52.91	y																	
159	2021-11-18 06:00:08	17.96	52.96	y																	
160	2021-11-18 07:00:07	20.37	53.8	y																	
161	2021-11-18 11:00:07	null	null																		
162	2021-11-18 12:00:07	null	null																		
163	2021-11-18 13:00:07	null	null																		
164	2021-11-18 14:00:07	null	null																		
165	2021-11-18 15:00:07	24.21	43.11	y																	
166	2021-11-18 16:00:07	21.21	49.23	y																	
167	2021-11-18 17:00:07	19.24	53.6	y																	
168	2021-11-18 18:00:07	null	null																		
169	2021-11-18 19:00:08	18.5	55.28	y																	
170	2021-11-18 20:00:07	18.18	55.21	y																	
171	2021-11-18 21:00:07	18.19	55.28	y																	
172	2021-11-18 22:00:07	18.65	54.59	y																	
173	2021-11-18 23:00:07	18.95	54.07	y																	
174	2021-11-19 00:00:07	19.12	53.72	y																	
175	2021-11-19 01:00:07	null	null																		
176	2021-11-19 02:00:07	19.59	51.04	y																	
177	2021-11-19 03:00:07	18.54	52.77	y																	
178	2021-11-19 11:00:07	18.83	53.6	y																	
179	2021-11-19 12:00:07	20.73	49.33	y																	
180	2021-11-19 13:00:07	21.16	48	y																	
181	2021-11-19 14:00:07	18.77	52.25	y																	
182	2021-11-19 15:00:07	19.07	51.77	y																	
183	2021-11-19 16:00:08	15.91	54.39	y																	
184	2021-11-19 17:00:08	13.43	55.56	y																	

Sheet1 | Selected: 155 rows, 1 column | Default | English (USA) | Average: ; Sum: 0 | 100%

Our file is ready for training AWS AI. At this stage we need this file to be downloaded from internet, so I create a public github repository and put that file into repository. Here is the repository link <https://github.com/mkhan-sfbu/AWS-IoT-Raspberry-Pi-Emulator-Sensor-HAT-Emulator-Predictive-Analysis>. Also the training file uploaded into that repository like as follows -

Now we have public downloadable link of our training data which we need later.

## Train AWS AI

To train AWS AI we need some space where our filtered train data need to store. Also we need machine to run train script, here we use SageMaker Notebook of AWS. First of all we need to create a S3 bucket to store our training data. As we created bucket already before for our previous homework, I'm not going to show in details here.

We created S3 bucket named “ee517week12hw2-traindata”. Now we are going to create SageMaker Notebook Instance and we are going to use it for training purpose.

To create SageMaker Notebook Instance first go to “SageMaker” but searching at the top search panel of AWS console. Then open “Notebook” drop down link from left menu and then click “Notebook Instances”. Please note, this instance means the virtual machine which managed by AWS. You can execute python code and any code by SSH login.

The screenshot shows the "Domains" page in the Amazon SageMaker console. On the left, there's a sidebar with various links like "Getting started", "Studio", "Studio Lab", "Canvas", "RStudio", "Domains", "SageMaker dashboard", "JumpStart", and "Notebook". The main content area is titled "Domains" and contains a table with one row. The table columns are Name, Id, Status, Created on, and Modified on. The single entry is "ee517week12hw2" with ID "d-ddgkicpcja9e", status "InService", created on "Dec 07, 2022 23:32 UTC", and modified on "Dec 07, 2022 23:38 UTC". There are "View", "Edit", and "Create domain" buttons at the top right of the table.

In that page you will find “Create Notebook Instance” link at top right corner, click on that link will land following page -

The screenshot shows the "Create notebook instance" page in the Amazon SageMaker console. The left sidebar has the same set of links as the previous screenshot. The main form is titled "Create notebook instance". It has two main sections: "Notebook instance settings" and "Permissions and encryption". In the "Notebook instance settings" section, fields include "Notebook Instance name" (set to "week12hw2notebook"), "Notebook Instance type" (set to "ml.t3.medium"), "Elastic Inference" (set to "none"), and "Platform identifier" (set to "Amazon Linux 2, Jupyter Lab 1"). Below these is a "Additional configuration" button. In the "Permissions and encryption" section, there's an "IAM role" dropdown set to "SageMaker-ShrkrData", a "Root access - optional" section with "Enable" selected, and an "Encryption key - optional" dropdown set to "No Custom Encryption". At the bottom, there are "Feedback", "Cookie preferences", and standard footer links.

Input name of notebook instance and select a IAM role, if you don't have any IAM role system will create a new one automatically for you. Just keep that role and keep all default value of this page and click "Create notebook instance" button from end of form.

After creating notebook instance we need to wait a bit until status said "InService" as bellow -

The screenshot shows the Amazon SageMaker console interface. On the left, there's a sidebar with various services like Studio, Studio Lab, Canvas, RStudio, Domains, SageMaker dashboard, JumpStart, and Notebook. The 'Notebook instances' section is selected. A specific notebook instance, 'week12hw2notebook', is shown in the main panel. Its details include:

- Name:** week12hw2notebook
- Status:** InService
- Notebook instance type:** ml.t3.medium
- Platform identifier:** Amazon Linux 2, Jupyter Lab 1 (notebook-a12-v1)
- ARN:** arn:aws:sagemaker:us-east-1:669425057253:notebook-instance/week12hw2notebook
- Creation time:** Dec 10, 2022 06:10 UTC
- Last updated:** Dec 10, 2022 06:13 UTC
- Elastic Inference:** -
- Volume Size:** 5GB EBS
- Minimum IMDS Version:** 2
- Lifecycle configuration:** -

Below this, there are sections for 'Git repositories' and 'Permissions and encryption', both currently empty. At the top right of the main panel, there are 'Delete', 'Stop', 'Open Jupyter' (which is highlighted), and 'Open JupyterLab' buttons.

After getting "InService" status we need to click "Open Jupyter" button from top right panel. This will open Jupyter control panel for this instance in a new tab as follows -

The screenshot shows the Jupyter control panel for the 'week12hw2notebook' instance. The top bar shows the URL 'week12hw2notebook.notebook.us-east-1.sagemaker.aws/tree'. Below the top bar, there's a navigation bar with 'Files', 'Running', 'Clusters', 'SageMaker Examples', and 'Conda'. A context menu is open over a file named 'Untitled.ipynb'. The menu items include:

- Upload
- New
- Create a new notebook with conda\_python3
- Other: Text File, Folder, Terminal

The main area shows a list of notebooks and files in the directory, including 'R', 'Sparkmagic (PySpark)', 'Sparkmagic (Spark)', 'Sparkmagic (SparkR)', 'conda\_amazonel\_mxnet\_p36', 'conda\_amazonel\_pytorch\_latest\_p37', 'conda\_mxnet\_p37', 'conda\_python3', 'conda\_pyl1', and 'conda\_tensorflow2\_p38'.

From this page we need to open new "conda\_python3" module to run our python code for training. We can find that link from "New" drop down link of top right corner. When we click this link, it will open a new Python code editor in a new tab. Basically it's a Python code executor by which we will train AWS AI and create model for our training data through. We are going to apply a bunch of Python code by typing, coping or pasting. Here we need to execute each code either by clicking "Run" button at the top bar or we can press "Shift+Enter". After execute each python command, system will print output (if any) at the bottom of that command line.

Some Python command applied and some as output at following screenshot -

```
import boto3
import re
from sagemaker import get_execution_role
import numpy as np
numerical processing
import pandas as pd
import matplotlib.pyplot as plt
visualizations
from IPython.display import Image
the notebook
from IPython.display import display
the notebook
from time import gmtime, strftime
models, endpoints, etc.
import sys
notebook
import math
import json
outputs
import os
names
import sagemaker
SDK provides many helper functions
from sagemaker.predictor import csv_serializer

role = get_execution_role()
bucket = 'ee517week12hw2-traindata'
```

```
In [3]: !curl https://raw.githubusercontent.com/mkhan-sfbu/AWS-IoT-Raspberry-Pi-Emulator-Sensor-HAT-Emulator-Predictive-Analysis/main/train-data.csv
<--2022-12-10 19:55:32-- https://raw.githubusercontent.com/mkhan-sfbu/AWS-IoT-Raspberry-Pi-Emulator-Sensor-HAT-Emulator-Predictive-Analysis/main/train-data.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.133, 185.199.110.133, 185.199.111.1
33...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7493 (7.3K) [text/plain]
Saving to: 'train-data.csv'

100%[=====] 7,493      --.-K/s   in 0s

2022-12-10 19:55:32 (45.4 MB/s) - 'train-data.csv' saved [7493/7493]
```

```
In [4]: data = pd.read_csv('./train-data.csv')
pd.set_option('display.max_columns', 500)    # Make sure we can see all of the columns
pd.set_option('display.max_rows', 20)
```

```
In [ ]:
```

Here is the python codes that we use to train our model -

```
import boto3
import re
from sagemaker import get_execution_role
import numpy as np
numerical processing
import pandas as pd
import matplotlib.pyplot as plt
visualizations
from IPython.display import Image
the notebook
from IPython.display import display
the notebook
from time import gmtime, strftime
models, endpoints, etc.
import sys
notebook
import math
import json
outputs
import os
names
import sagemaker
SDK provides many helper functions
from sagemaker.predictor import csv_serializer

role = get_execution_role()
bucket = 'ee517week12hw2-traindata'
```

# For matrix operations and numerical processing  
# For munging tabular data  
# For charts and visualizations  
# For displaying images in the notebook  
# For displaying outputs in the notebook  
# For labeling SageMaker models, endpoints, etc.  
# For writing outputs to notebook  
# For ceiling function  
# For parsing hosting outputs  
# For manipulating filepath names  
# Amazon SageMaker's Python SDK provides many helper functions

```
prefix = 'notebookprefix'
```

We need our training CSV file which we stored into github! Following command will download that file from github -

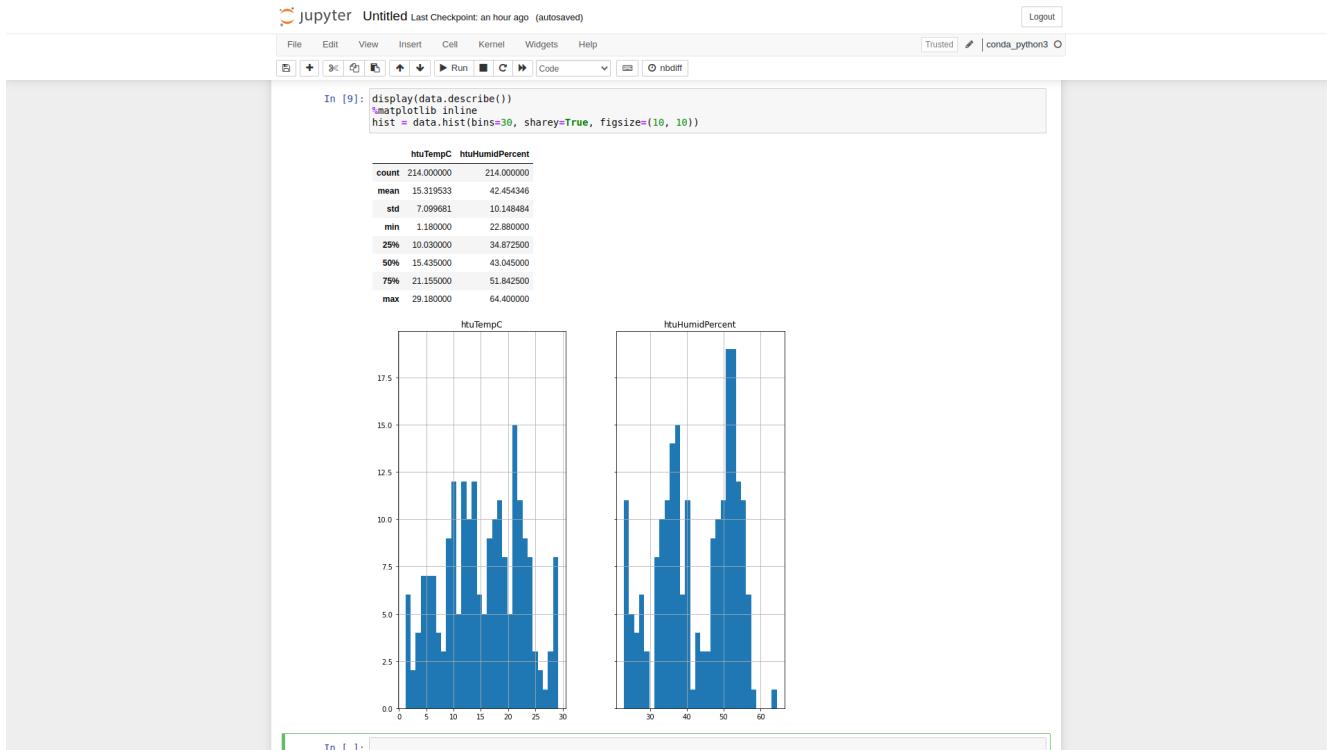
```
!wget https://raw.githubusercontent.com/mkhan-sfbu/AWS-IoT-Raspberry-Pi-Emulator-Sensor-HAT-Emulator-Predictive-Analysis/main/train-data.csv
```

Now we are going to use that downloaded file by loading through pandas.

```
data = pd.read_csv('./train-data.csv')
pd.set_option('display.max_columns', 500)      # Make sure we can see all of
the columns
pd.set_option('display.max_rows', 20)
```

We can display our data by using plot command as follows -

```
display(data.describe())
%matplotlib inline
hist = data.hist(bins=30, sharey=True, figsize=(10, 10))
```

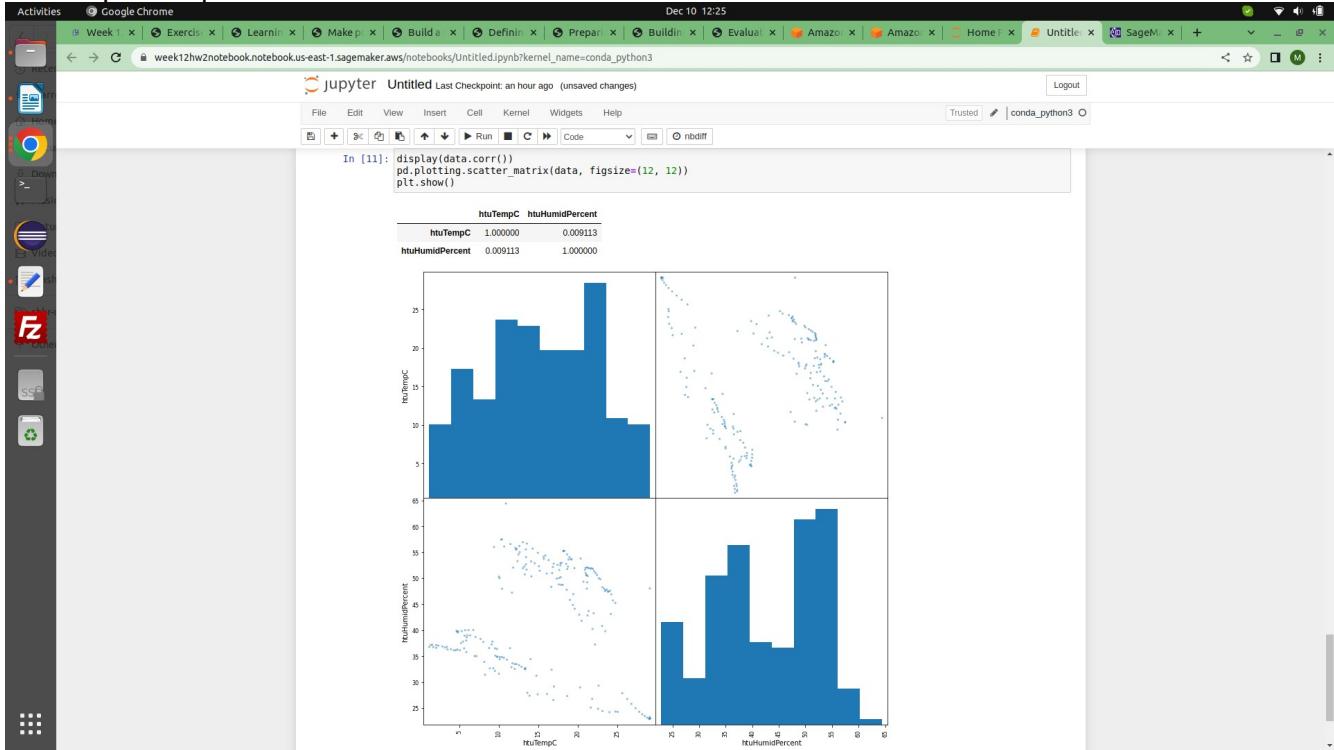


By placing our data in a scatter plot, we essentially compare each feature in our dataset to itself and we're given an idea of how strong the correlation is between features. This only works for numeric features, hence why you won't be seeing our target variable in this plot.

To show our data in a scatter plot we used following commands -

```
display(data.corr())
pd.plotting.scatter_matrix(data, figsize=(12, 12))
plt.show()
```

Scatter plot output as follows -



So far we just view our data to ensure that we have valid relation between data. Now we are going to train AWS AI and create model based on our data. First we convert our data in training format then we split it into three data set randomly which are train data, validation data and test data. As our data is not real we may get error while validation later on. Though I'm showing the process how one can split data for specific purpose. Here is the commands to apply! -

```
model_data = pd.get_dummies(data) # pandas read raw data and prepared
train_data, validation_data, test_data = np.split(model_data.sample(frac=1,
random_state=1729), [int(0.7 * len(model_data)), int(0.9 * len(model_data))])
```

Now we are going to store train and validation data. Please note we need to mention target variable in the first column. Here we target/predict “watering” column whee value is either “y” or “n” so final target column is “watering\_y”. Following command will store our data into our desired file name -

```
# Prep the training set for the XGBoost algorithm - the target variable must
# be the first column and store the dataset as a CSV
pd.concat([train_data['watering_y'], train_data.drop(['watering_n',
'watering_y'], axis=1)], axis=1).to_csv('train.csv', index=False,
header=False)
# Same as above for the validation set
```

```
pd.concat([validation_data['watering_y'], validation_data.drop(['watering_n', 'watering_y'], axis=1)], axis=1).to_csv('validation.csv', index=False, header=False)
```

Now we are going to store train and validation data into S3 bucket. So, our AI also can get those file easily when required. We will get help of boto3 project to communicate between our Python script and AWS S3 bucket. As previous we already set our bucket name, we are going to use here. Also we set prefix, so system will create folder into S3 bucket and will put those file into that folder.

```
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'train/train.csv')).upload_file('train.csv')
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'validation/validation.csv')).upload_file('validation.csv')
```

At this stage we are going to open SageMaker session to train model with out data by using following commands -

```
sess = sagemaker.Session()

xgb = sagemaker.estimator.Estimator(container,
                                      role,
                                      train_instance_count=1,
                                      train_instance_type='ml.m4.xlarge',
                                      output_path =
's3://{}{}'.format(bucket, prefix),
                                      sagemaker_session=sess)
xgb.set_hyperparameters(max_depth=5,
                       eta=0.2,
                       gamma=4,
                       min_child_weight=6,
                       subsample=0.8,
                       silent=0,
                       objective='binary:logistic',
                       num_round=100)
xgb.fit({'train': s3_input_train, 'validation': s3_input_validation})
```

Here is the output of above codes -

Activities Google Chrome Dec 11 12:40

Inbox - mkhahan@student... | Grades for MD SHAHAD... | mkhahan-sfbu/AWS-IoT-Ra... | Amazon SageMaker | Home Page - Select or cre... | week12hw3 - Jupyter Note... +

week12hw2notebook.notebooks-east-1.sagemakeraws/notebooks/week12hw3.ipynb

Jupyter week12hw3 Last Checkpoint: Yesterday at 11:26 AM (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted conda/python3 O

In [17]: `model_data = pd.get_dummies(data)`

In [18]: `train_data, validation_data, test_data = np.split(model_data.sample(frac=1, random_state=1729), [int(0.7 * len(model_data)), len(model_data) * 0.15, len(model_data) * 0.15])`  
# Prep the training set for the XGBoost algorithm - the target variable must be the first column and store the data:  
`pd.concat([train_data['watering_n'], train_data.drop(['watering_n', 'watering_y'], axis=1)], axis=1).to_csv('train.csv')`  
# Same as above for the validation set  
`pd.concat([validation_data['watering_y'], validation_data.drop(['watering_n', 'watering_y'], axis=1)], axis=1).to_csv('validation.csv')`

In [19]: `from sagemaker.amazon.amazon_estimator import get_image_uri`  
container = get\_image\_uri(botocore.Session().region\_name, 'xgboost')  
The method get\_image\_uri has been renamed in sagemaker>=2.  
See: <https://sagemaker.readthedocs.io/en/stable/v2.html> for details.

In [20]: `s3_input_train = sagemaker.inputs.TrainingInput(s3_data='s3://{}//train'.format(bucket, prefix), content_type='csv')`  
`s3_input_validation = sagemaker.inputs.TrainingInput(s3_data='s3://{}//validation'.format(bucket, prefix), content_type='csv')`

In [21]: `sess = sagemaker.Session()`  
`xgb = sagemaker.estimator.Estimator(container,`  
role,  
train\_instance\_count=1,  
train\_instance\_type='ml.m4.xlarge',  
output\_path='s3://{}//output'.format(bucket, prefix),  
sagemaker\_session=sess)  
`xgb.set_hyperparameters(max_depth=5,`  
eta=0.2,  
gamma=4,  
min\_child\_weight=6,  
subsample=0.8,  
silent=0,  
objective='binary:logistic',  
num\_round=100)

train\_instance\_count has been renamed in sagemaker>=2.  
See: <https://sagemaker.readthedocs.io/en/stable/v2.html> for details.  
train\_instance\_type has been renamed in sagemaker>=2.  
See: <https://sagemaker.readthedocs.io/en/stable/v2.html> for details.

In [22]: `xgb.fit({'train': s3_input_train, 'validation': s3_input_validation})`

2022-12-11 06:51:48 Starting - Starting the training job...

After finish execution of above code the output will show as follows. Please note that on “Ln [22]” you will see like “Ln [\*]” while executing code. After finish complete it shows the line number. This is important that we need to wait till complete. After code execution finish we will apply further commands.

Activities Google Chrome Dec 11 13:15

Inbox - mghan@student... Grades for MD SHAHAD... mghan-sfbu/AWS-IoT-Ras... Amazon SageMaker Home Page - Select or create week12hw3 - Jupyter Notebooks

week12hw3notebook.notebook.us-east-1.sagemaker.aws/notebooks/week12hw3.ipynb

Jupyter week12hw3 Last Checkpoint: Yesterday at 11:26 AM (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | conda\_python3 O

In [22]: `xgb.fit('train': s3_input_train, 'validation': s3_input_validation)`

[06:55:03] src/tree/updater prune.cc:74: tree pruning end, 1 roots, 0 extra nodes, 2 pruned nodes, max\_depth=0  
[94]#01ltrain-error:0.127517#01lvalidation-error:0.139535  
[06:55:03] src/tree/updater prune.cc:74: tree pruning end, 1 roots, 0 extra nodes, 2 pruned nodes, max\_depth=0  
[95]#01ltrain-error:0.127517#01lvalidation-error:0.139535  
[06:55:03] src/tree/updater prune.cc:74: tree pruning end, 1 roots, 0 extra nodes, 2 pruned nodes, max\_depth=0  
[96]#01ltrain-error:0.127517#01lvalidation-error:0.139535  
[06:55:03] src/tree/updater prune.cc:74: tree pruning end, 1 roots, 0 extra nodes, 2 pruned nodes, max\_depth=0  
[97]#01ltrain-error:0.127517#01lvalidation-error:0.139535  
[06:55:03] src/tree/updater prune.cc:74: tree pruning end, 1 roots, 0 extra nodes, 2 pruned nodes, max\_depth=0  
[98]#01ltrain-error:0.127517#01lvalidation-error:0.139535  
[06:55:03] src/tree/updater prune.cc:74: tree pruning end, 1 roots, 0 extra nodes, 2 pruned nodes, max\_depth=0  
[99]#01ltrain-error:0.127517#01lvalidation-error:0.139535

2022-12-11 06:55:18 Training - Training image download completed. Training in progress.  
2022-12-11 06:55:18 Uploading - Uploading generated training model  
2022-12-11 06:55:18 Completed - Training job completed  
Training seconds: 65  
Billable seconds: 65

In [23]: `xgb_predictor = xgb.deploy(initial_instance_count=1, instance_type='ml.t2.medium')`  
xgb\_predictor.serializer = csv\_serializer  
-----!

In [24]: `def predict(data, rows=500):  
 split_array = np.array_split(data, int(data.shape[0] / float(rows)) + 1)  
 predictions = ''  
 for array in split_array:  
 predictions += ','.join([predictions, xgb_predictor.predict(array).decode('utf-8')])  
  
 return np.fromstring(predictions[1:], sep=',')`  
predictions = predict(test\_data.drop(['watering\_n', 'watering\_y'], axis=1).to\_numpy())  
The csv serializer has been renamed in sagemakers2.  
See: <https://sagemaker.readthedocs.io/en/stable/v2.html> for details.

In [25]: `pd.crosstab(index=test_data['watering_y'], columns=np.round(predictions), rownames=['actuals'], colnames=['predicted'])`

Out[25]: `predictions 0.0 1.0`  
actuals  

	0	1
0	3	3
1	0	16

After finish our training we need to deploy our model, so that we can get predictive result for our IoT devices. Please note, after finishing deployment one endpoint for this deployment will create

automatically at SageMaker control panel, which we need later and will retrieve later. Here is the deploy code -

```
xgb_predictor = xgb.deploy(initial_instance_count=1,  
instance_type='ml.t2.medium')
```

This deploy also need some time, we have to wait to apply further command. After finishing deployment we can check our deployment through our test data by following commands -

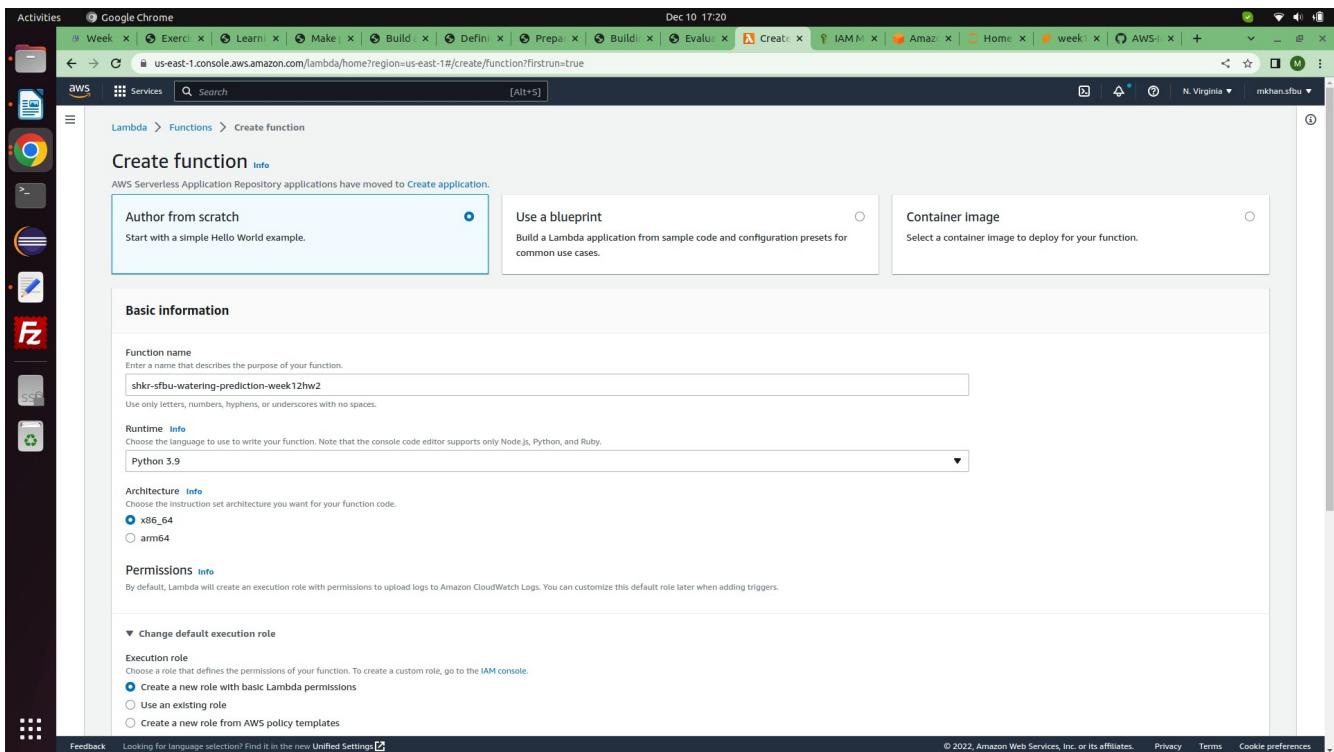
```
xgb_predictor.serializer = csv_serializer  
def predict(data, rows=500):  
    split_array = np.array_split(data, int(data.shape[0] / float(rows) + 1))  
    predictions = ''  
    for array in split_array:  
        predictions = ','.join([predictions,  
xgb_predictor.predict(array).decode('utf-8')])  
    return np.fromstring(predictions[1:], sep=',')  
  
predictions = predict(test_data.drop(['watering_n', 'watering_y'],  
axis=1).to_numpy())  
  
pd.crosstab(index=test_data['watering_y'], columns=np.round(predictions),  
rownames=['actuals'], colnames=['predictions'])
```

Output of above code is shown in above screenshot. That's all we are ready to get data into our IoT devices, but for this we need to go through some process in AWS.

## Prepare AWS to serve data

We do ready our model for prediction, now we need to get that predictive data into our IoT devices. To do this we need to create a lambda function of AWS which responsible for grab data from our SageMaker model. Then we need to create a REST API service at AWS which will execute that lambda function and send the result as a response of REST API request.

To create lambda function search lambda at the top search bar of AWS consol and then create lambda dashboard, at that dashboard you will find “Create function” link. After clicking that link it will open following page. Here just input name of the function and select runtime as “Python 3.9”. Select your architecture, mine is “x86\_64” and select “Create a new role with basic Lambda permissions” into execution role section. Leave all other as is and click “Create” button from bottom of page.

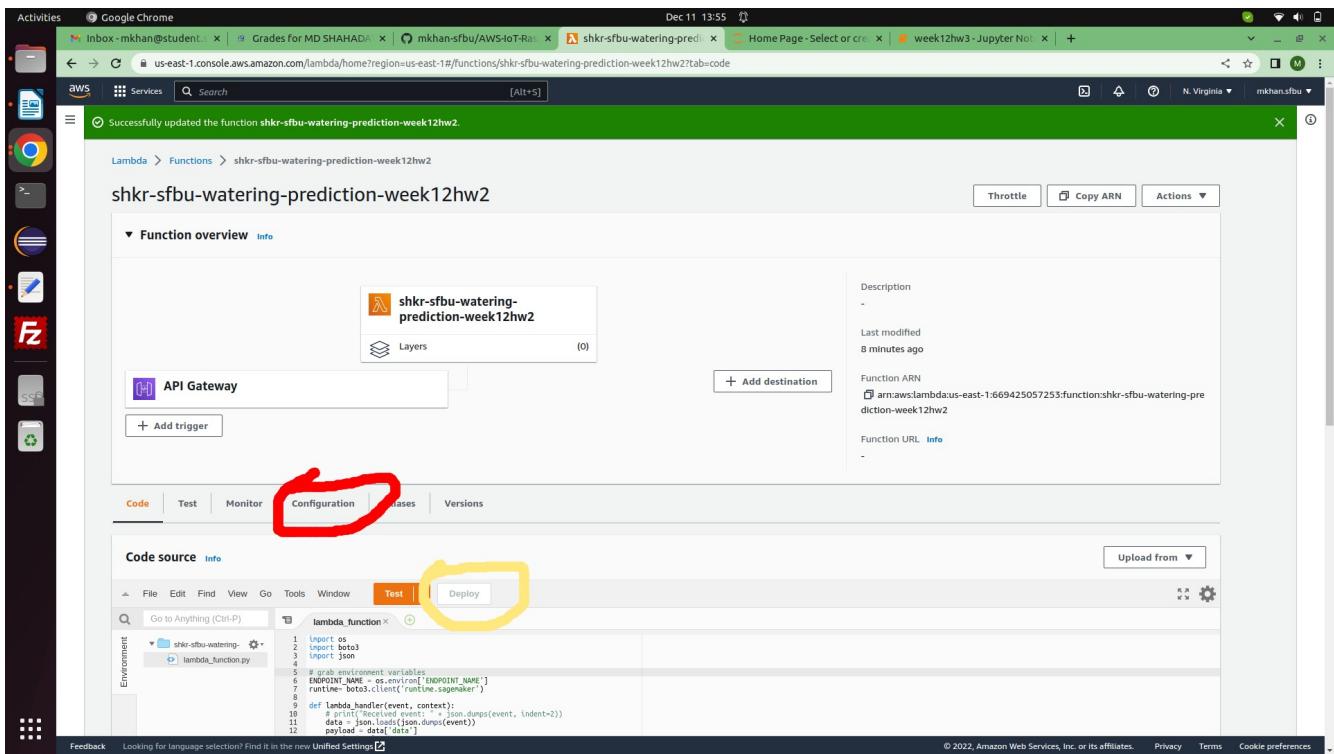


Dashboard of Lambda function we created will show and with code editor to put following code.

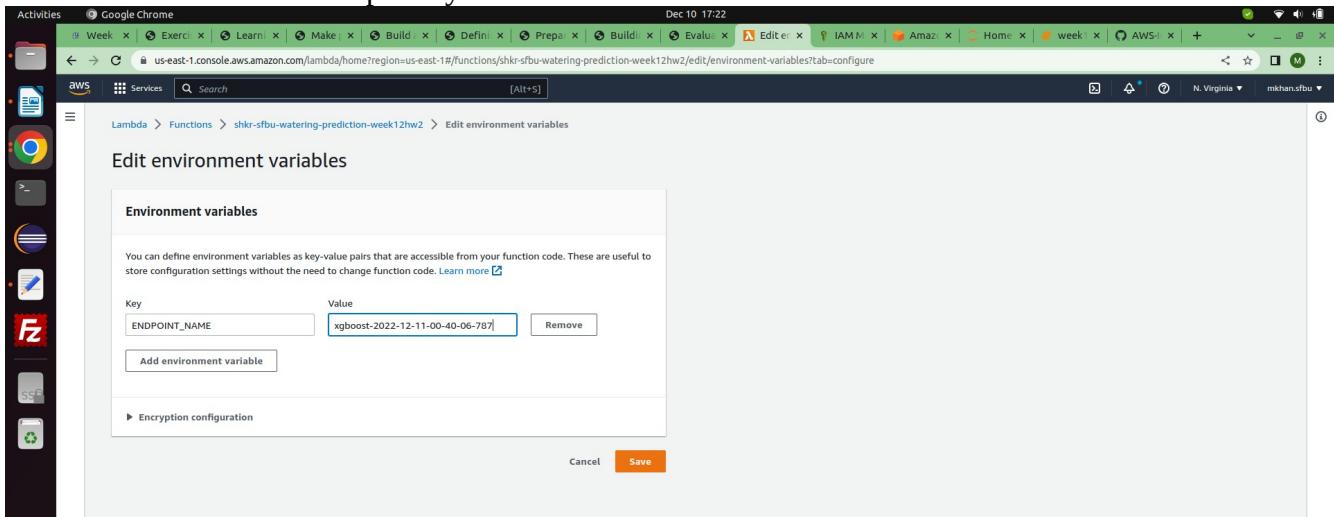
```
import os
import boto3
import json
ENDPOINT_NAME = os.environ['ENDPOINT_NAME']
runtime= boto3.client('runtime.sagemaker')
def lambda_handler(event, context):
    data = json.loads(json.dumps(event))
    response = runtime.invoke_endpoint(EndpointName=ENDPOINT_NAME,
                                       ContentType='text/csv', Body=data['data'])
    result = json.loads(response['Body'].read().decode())
    return 'W' if float(result) > 0.5 else 'X'
```

It's a simple code, we get our SageMaker model endpoint name from environment variable. Into `lambda_handler` method we invoke our endpoint by providing payload and after that we load model response into json module and decode it for our result. Here as our data is not good we get result between 0 ~ 1. So, I response "W" means need watering if we have result 50% positive otherwise returns "X". This 50% I assumed from based on my data, anyone can play with this percentage.

Here is the interface screenshot. Note that we need to save our code by clicking "File > Save" at editor. After saving code we need to deploy by clicking "Deploy" (will active when we change code) button at editor menu bar (yellow marked).



Now two major things is how to set environment variable that required to execute this command. To set environment variable click “Configuration” link (red marked in above screenshot) then click “Environment variables” link at the left side menu. Into that page click “Edit” button from right top corner of that section. After put key and value click “Save” button.



Before going environment variable settings we need SageMaker deployment endpoint name. To get that name we need to go SageMaker dashboard and then need to open “Inference” link from left menu and click on “Endpoints” from the list. Here we will get all available endpoints of our project. Here we found our endpoint that automatically generated from deployment code.

From endpoints list we can easily get endpoint name that need to use into environment variable above.

Lambda function completed, now we need to setup REST API. To do this search “rest” at the top search bar of consol you will get “API Gateway” link, click that link will land following dashboard -

Clicking on “Build” button of REST API will open following page. Here we need to select “REST” and need to select “New API”. Also we need to provide a name of our API and need to select Endpoint Type which is “Regional” then we need to click “Create API” button at bottom right corner.

Choose the protocol  
Select whether you would like to create a REST API or a WebSocket API.  
REST  WebSocket

Create new API  
In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.  
New API  Import from Swagger or Open API 3  Example API

Settings  
Choose a friendly name and description for your API.

API name\* shkr-sfbu-week12hw2-watering  
Description  
Endpoint Type Regional

\* Required Create API

After creating API we need to click on “Action” drop down list and there we will find “Create Resource” link. Clicking on “Create Resource” link we will land on following page. Here we need to provide new resource name and need to click “Create Resource” link at bottom right corner.

Activities Google Chrome Dec 10 17:29 us-east-1.console.aws.amazon.com/api/gateway/home?region=us-east-1#/apis/ytoyerdd09d/resources/w59qesu295/create N. Virginia mkhan.sfbu Show all hints ?

APIs > shkr-sfbu-week12hw2-watering (ytoyerdd09d) > Resources > /w59qesu295 > Create

Actions New Child Resource

Configure as proxy resource

Resource Name\* week12hw2watering

Resource Path\* /week12hw2watering

You can add path parameters using brackets. For example, the resource path `{username}` represents a path parameter called 'username'. Configuring `/proxy{}` as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to `/foo`. To handle requests to `/`, add a new ANY method on the `/` resource.

Enable API Gateway CORS

\* Required Cancel Create Resource

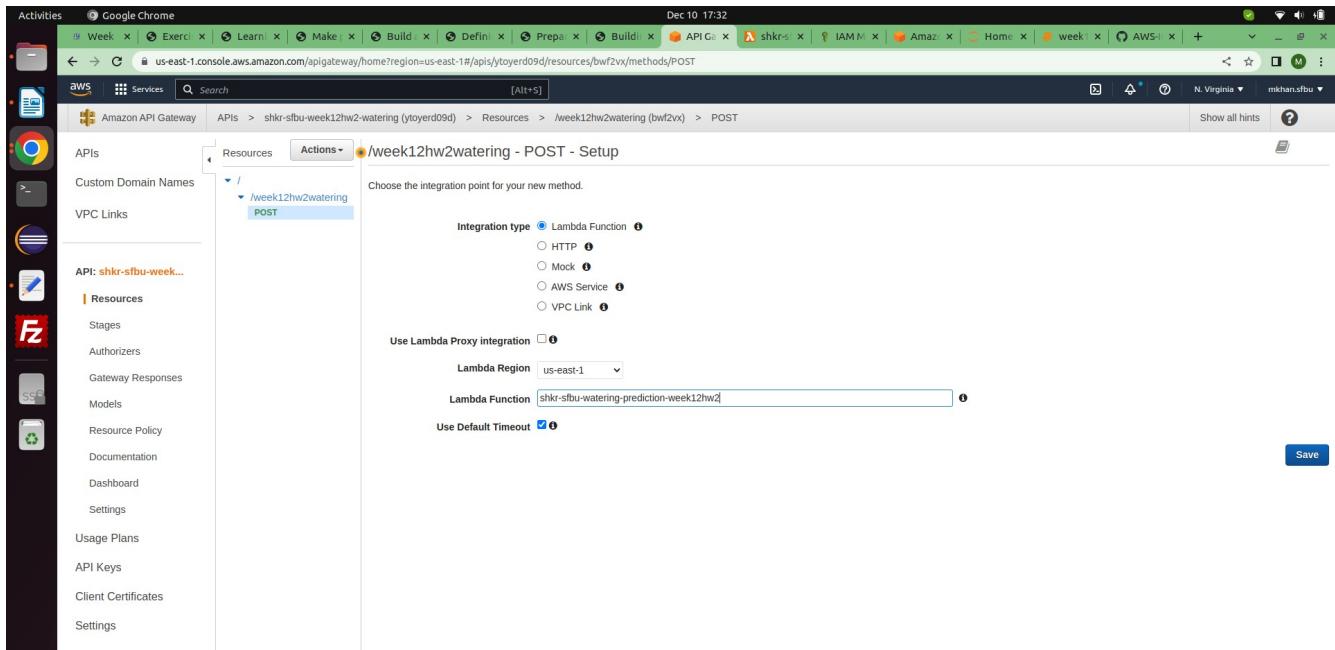
After creating resource, we need to create method (POST). Again we need to click on “Action” drop down button and we will find “Create Method” link. Please note, newly created resource will automatically selected but if it is not then, we have to select our resource at the middle section.

The screenshot shows the AWS API Gateway console. On the left, there's a sidebar with various navigation options like APIs, Custom Domain Names, VPC Links, and others. The main area shows an API named "shkr-sfbu-week...". Under the "Resources" section, a specific resource path is selected: "/week12hw2watering". A context menu is open over this path, with the "Actions" button highlighted. The menu has two sections: "RESOURCE ACTIONS" which includes "Create Method", "Create Resource", "Enable CORS", "Edit Resource Documentation", and "Delete Resource"; and "API ACTIONS" which includes "Deploy API", "Import API", "Edit API Documentation", and "Delete API". Below the menu, a message says "No methods defined for the resource."

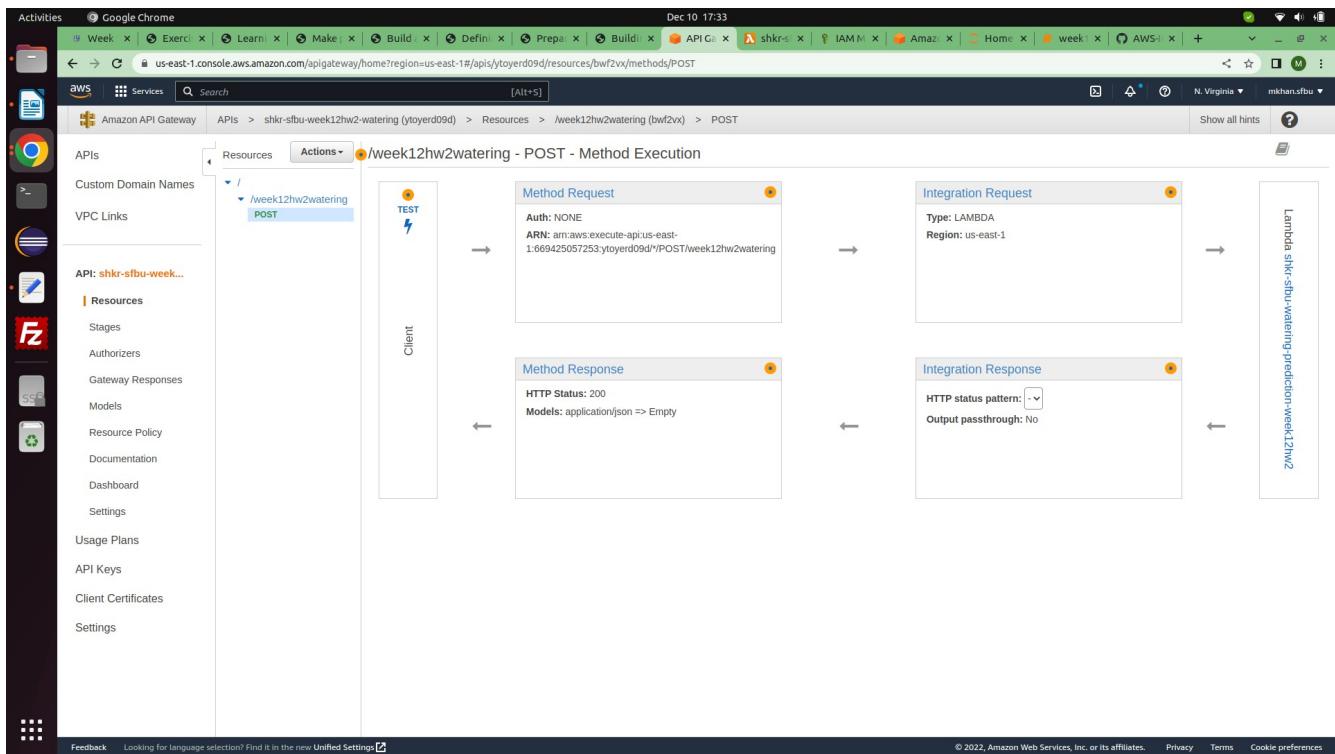
When we click “Create Method” we need to select which type of method we are going to create from drop down list as shown bellow. We are going to select “POST” method, cause we are going to query for our prediction result through POST method.

This screenshot is similar to the previous one but focuses on the "Create Method" dropdown. The dropdown menu is open, showing a list of HTTP methods: ANY, DELETE, GET, HEAD, OPTIONS, PATCH, POST, and PUT. The "POST" option is highlighted with a blue selection bar. The rest of the interface is identical to the first screenshot, showing the API structure and the message "No methods defined for the resource."

When we select “POST” method one page open to create this method. In this page we need to tell that we are going to execute “Lambda Functions” on this method invoke. And we need to provide that Lambda Function name. Please keep region and timeout setting as default. Finally click “Save”

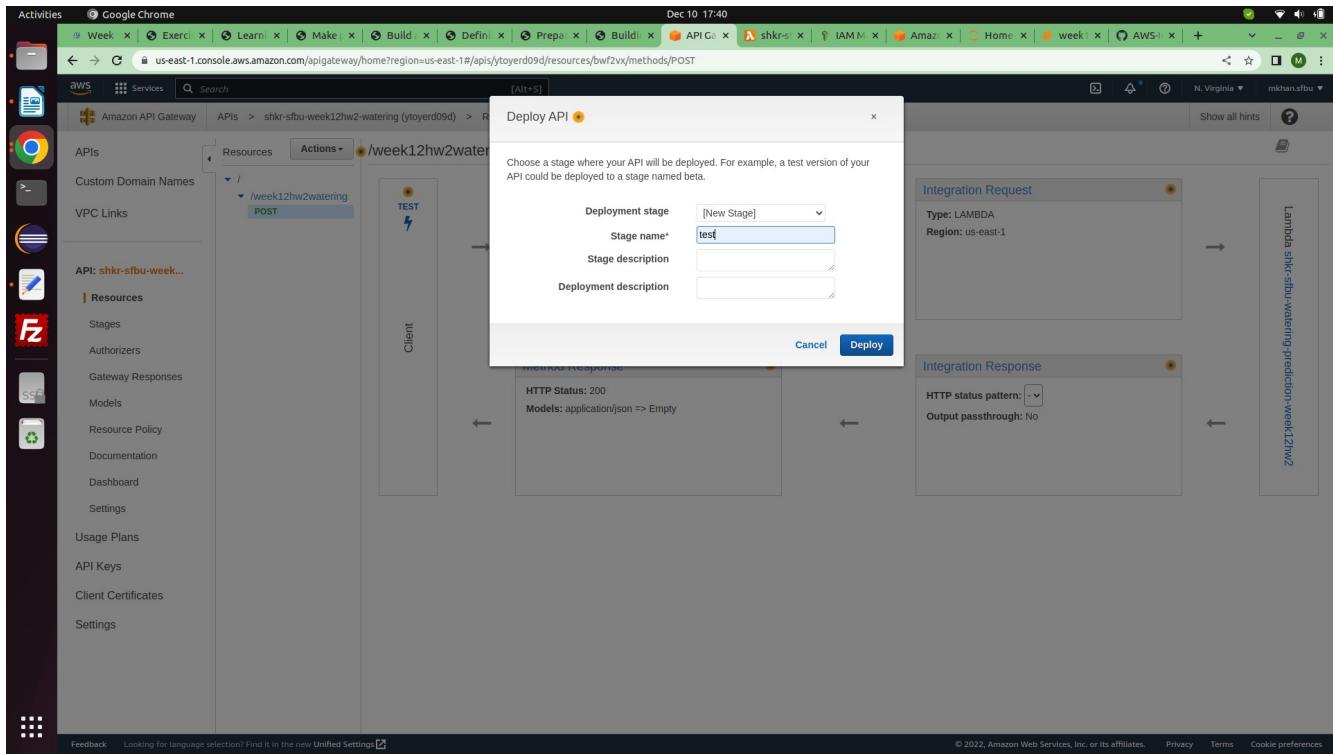


Following page will appear after finish our post method creation. At this stage we need to deploy our API. To do this click “Action” drop down again, there “Deploy API” link will found.

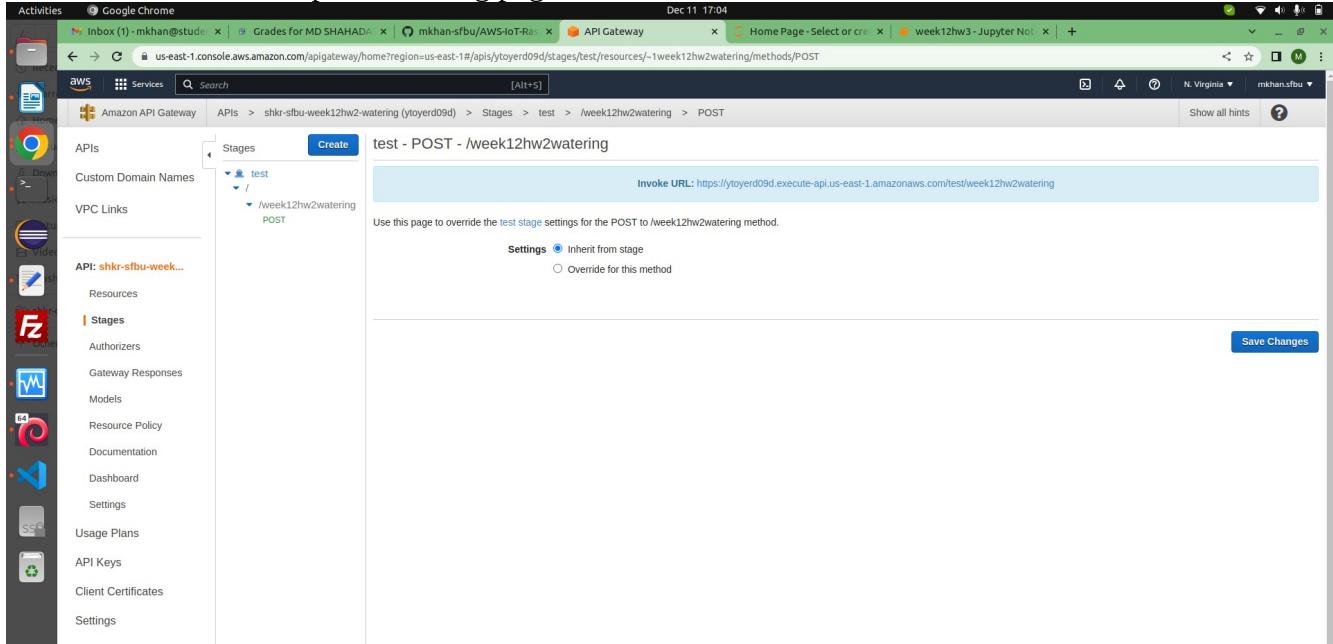


When we click “Deploy API” on popup box will appear from where we need to select “Deployment stage” as “[New Stage]” and then we need to provide stage name, here we provide “test” as its our test

deployment. And then click “Deploy” will deploy this API and we can start querying the API for predictive result.



That's all from AWS configuration, we deployed API to get predictive result. Now we are going to query this API from our Raspberry Pi with SensorHAT data. To query to REST API we need API endpoint URL. To get the URL click on “Stages” link from left menu and select “POST” method from middle bar which will open following page -



From above page we need to copy/save “Invoke URL” to which we need to query from our device.

## Retrieve predictive data into Raspberry Pi

As we need temperature and humidity data to query AWS for predictive result we are going to use SanseHAT emulator to get those data. Following Python code will query to AWS REST API and show appropriate message to user based on result which is if user need watering or not based on current temperature and humidity data.

```
from sense_emu import SenseHat
import signal
import sys
import requests
import json
from decimal import Decimal

def round_float(v, ndigits=0, rt_str=False):
    d = Decimal(v)
    v_str = ("{0:.%sf}" % ndigits).format(round(d, ndigits))
    if rt_str:
        return v_str
    return Decimal(v_str)

sense = SenseHat()

def destroyAndRelease():
    print("Ctrl+C detected")
    print("    ", end="\r", flush=True)

def handleExit(signum, frame):
    destroyAndRelease()
    sys.exit(1)

signal.signal(signal.SIGINT, handleExit)

prvTempVal=curTempVal=round_float(sense.temp)
prvHumdVal=curHumdVal=round_float(sense.humidity)

awsApiUrl =
'https://ytoyerd09d.execute-api.us-east-1.amazonaws.com/test/week12hw2watering
'

while True:
    curTempVal=round_float(sense.temp)
    curHumdVal=round_float(sense.humidity)
    if prvTempVal!=curTempVal or prvHumdVal!=curHumdVal:
        prvTempVal=curTempVal
        prvHumdVal=curHumdVal
        tempValue = round_float(float(curTempVal) / 2.5 + 16, 2)
        humidityValue = round_float(64 * float(curHumdVal) / 100, 2)
        print("Temperature or Humidity Changed!", curTempVal, curHumdVal)
        payload={'data': str(tempValue)+', '+str(abs(humidityValue))}

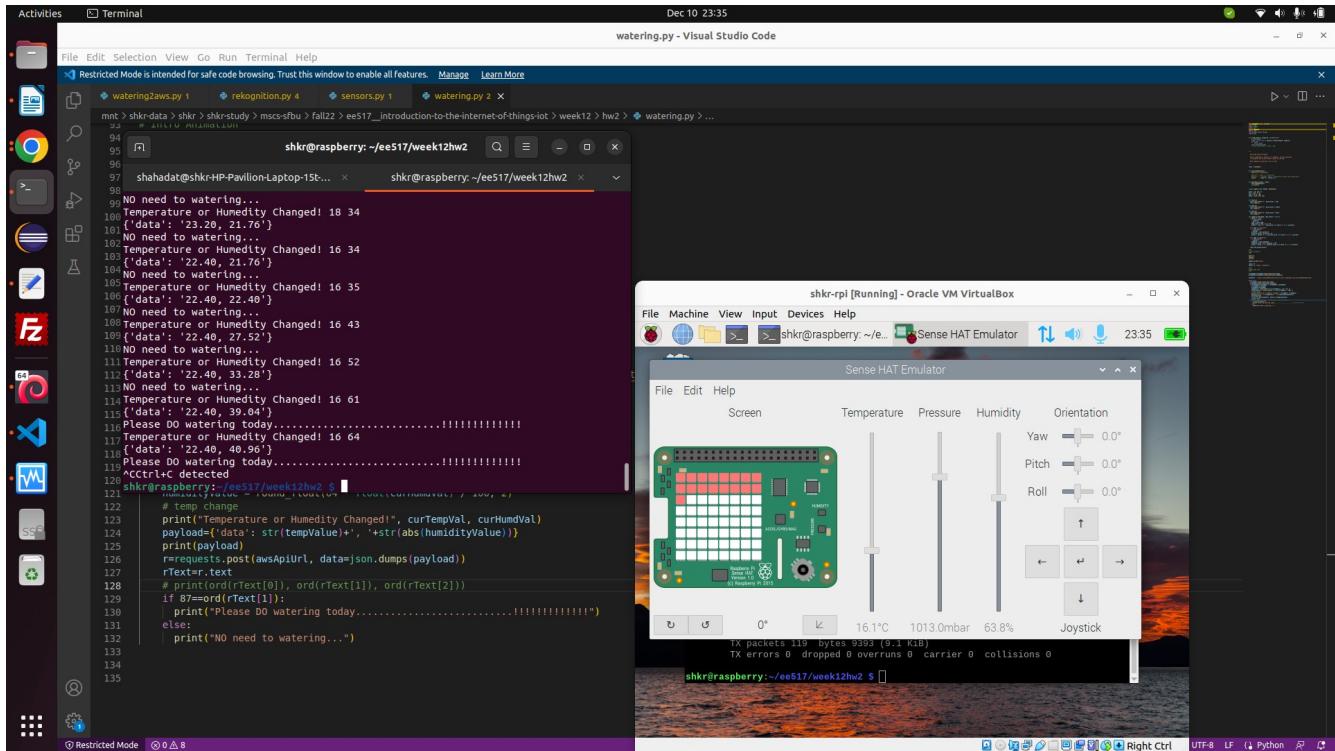
        print(payload)
        r=requests.post(awsApiUrl, data=json.dumps(payload))
        rText=r.text
        if 87==ord(rText[1]):
```

```

        print("Please DO watering
today.....!!!!!!")
else:
    print("NO need to watering...")

```

Above code are pretty straight forward, here we check temperature and humidity data change in a infinity while loop. Before execute that loop we setup “Ctrl+C” trap, so if user press “Ctrl+C” our program will terminate. And we query to our API by Python’s “requests” module. If we get result we comparing and show user like “No need to watering...”. Here is our program in action -



Code Repository: <https://github.com/mkhan-sfbu/AWS-IoT-Raspberry-Pi-Emulator-Sensor-HAT-Emulator-Predictive-Analysis>

Thank You